



# Digital Government Academy Course: Enterprise Content Management

Presented by Interwoven, Inc, for the State of Washington

# Today's Agenda

- ◆ Today we'll continue our exploration of Enterprise Content Management in the State of Washington

- ◆ Today's topics:

- **Develop-and-Deploy** →
- TeamSite in the Enterprise
- Branching Structures

- ◆ **Objectives:**

- The ECM development environment
- How deployment works
- The ROI of the TeamSite/OpenDeploy system
- Reuse of content via deployment

- ◆ **Action planning:**

- ◆ Comparing your expected ROI for implementing ECM
- Identifying deployment planning issues
- Identifying possible reuse scenarios during deployment

# Today's Agenda

- ◆ Today we'll continue our exploration of Enterprise Content Management in the State of Washington

- ◆ Today's topics:

- Develop-and-Deploy
- **TeamSite in the Enterprise** →
- Branching Structures

## ◆ Objectives:

- The architecture of the TeamSite server
- How the TeamSite server relates to other enterprise components

## ◆ Action planning:

- Discussing agencies' plans for integrating TeamSite with other organizational servers

# Today's Agenda

- ◆ Today we'll continue our exploration of Enterprise Content Management in the State of Washington

- ◆ Today's topics:

- Develop-and-Deploy
- TeamSite in the Enterprise
- **Branching Structures**



- ◆ **Objectives:**

- TeamSite branching patterns
- Branching and virtualization issues
- Locking models
- Using branches to implement re-use and sharing

- ◆ **Action planning:**

- ◆ Designing your group's branching model

# Seminar Schedule

## ◆ Day 1: December 4, 2002

- Course Kickoff
- ECM
- TeamSite Templating

## ◆ Day 2: December 11, 2002

- Develop-and-Deploy
- TeamSite in the Enterprise
- Branching Structures

## ◆ Day 3: December 18, 2002

- TeamSite Security
- Workflow Scenarios
- Designing a Workflow

## ◆ Day 4: January 8, 2003

- Designing Data Capture Forms
- Designing Presentation Templates
- Re-use via Templating

## ◆ Day 5: January 15, 2003

- TeamSite and Metadata
- Finding your Assets
- Supporting Personalization
- Course Summary



## The Develop-and-Deploy Model

# Topic Objectives

- ◆ In this topic, we'll cover:
  - The ECM development environment
  - How deployment works
  - The ROI of the TeamSite/OpenDeploy system
  - Reuse of content via deployment
- ◆ The action planning at the end of this topic will be:
  - Comparing your expected ROI for implementing ECM
  - Identifying deployment planning issues
  - Identifying possible reuse scenarios during deployment

# The Develop-and-Deploy Model

- ◆ When you work in an ECM, you work in a 2-stage process:
  - First content is developed (which includes writing, testing, reviewing, etc)
  - Then the finished content is delivered
- ◆ Development happens inside the **development environment**
  - For instance, **TeamSite**
- ◆ The finished content is delivered to the **production environment**
  - For instance, a **web server**
- ◆ By separating the two environments we gain stability and control

# TeamSite Development Environment: Advantages

- ◆ Stable work environment, separate from production
- ◆ Secure
- ◆ Version-controlled
- ◆ Simulates production environment
- ◆ Encourages collaboration
- ◆ Provides process and development simplification tools

# Deployment

- ◆ Deployment is the ECM term for the movement of content from the development environment to the production environment
- ◆ We normally accomplish this with Interwoven OpenDeploy
- ◆ Besides simply moving files, deployment involves:
  - Selecting required content
  - Security
  - Filtering
  - Permission setting
  - Performing automated tasks related to deployment

# Benefits of ECM Develop-and-Deploy

- ◆ Time savings
- ◆ Cost savings
- ◆ Increased control
- ◆ Increased stability
- ◆ Recoverability

# Example: Process Improvement



## Before TeamSite

Total Time = **135 min**

10. Deployed to production	10 min	Find file(s) > copy file(s) > paste file(s)
9. Doc owner does final acceptance	5 min	
8. Web Admin checks for completeness and/or errors in doc or code	20 min	
7. Manager approves moving document to production environment	10 min	
6. Doc owner now reviews Web doc in context	5 min	
5. Web admin manually moves file from server to development/QA	5 min	
4. Doc owner completes checklist that provides doc info	10 min	Checklist provides detailed instructions for QA of doc
3. Doc owner submits version control request through Lotus Notes DB on dev site & copies file to server	10 min	Notifies Web Server Admin of any pending changes
2. Doc owner creates/modifies Web doc	45 min	
1. Doc owner searches live site for latest version of doc to edit & downloads file to desktop	15 min	Search for html, JSP, graphic, Lotus Notes, etc...

## With TeamSite

Total Time = **69 min**

5. Web admin deploys content to production, upon final approval	1 min	Single click deployment
5. Doc owner sends "ok to deploy" to Web admin	2 min	
4. Within a workflow, manager receives QA request via email, makes comments or approves	10 min	
3. Doc Owner promotes to Staging	1 min	
2. Doc owner creates/modifies Web doc & virtualizes within workarea	45 min	<ul style="list-style-type: none"> <li>- Created in workarea, other people can work on asset simultaneously</li> <li>- Tested in-context of entire site, links and code included</li> </ul>
1. Doc owner logs into TS and opens file in workarea	10 min	

## Key Stats:

- Average time to **create** content without IWOV was 125 minutes and with IWOV was 68 minutes, taken from interview with <person 1> (see previous slide). Assume similar process across groups
- Assume 500 sites, based on response from <person 2>. This was assumed to be a conservative estimate
- Assumed each site created or updated 1,5, or 10 files a week. Some site will update more than 100x per week and others will be static.

10 files/week/site  
*High Estimate*

Reduce Development Costs	Current	With IWOV	Savings
Number of content changes each month	20,000	20,000	
Average minutes to develop content before it gets deployed	125	68	57
Average hourly fully burdened employee cost	\$36.06	\$36.06	
<b>Annual content development costs</b>	<b>\$18,028,846</b>	<b>\$9,807,692</b>	<b>\$8,221,154</b>

5 files/week/site  
*Mid Estimate*

Reduce Development Costs	Current	With IWOV	Savings
Number of content pages created each month	10,000	10,000	
Average minutes to develop content before it gets deployed	125	68	57
Average hourly fully burdened employee cost	\$36.06	\$36.06	
<b>Annual content development costs</b>	<b>\$9,014,423</b>	<b>\$4,903,846</b>	<b>\$4,110,577</b>

1 file/week/site  
*Low Estimate*

Reduce Development Costs	Current	With IWOV	Savings
Number of content pages created each month	2,000	2,000	
Average minutes to develop content before it gets deployed	125	68	57
Average hourly fully burdened employee cost	\$36.06	\$36.06	
<b>Annual content development costs</b>	<b>\$1,802,885</b>	<b>\$980,769</b>	<b>\$822,115</b>

# ROI: Content Deployment

## Key Stats:

- Average time to **deploy** content ranged from 30 minutes on average for <person 2> to 10 minutes for <name>
- Number of deployments per month based on 500 sites deploying on average of 1x per week each
- Average fully burdened employee cost based on \$75,000 per year

20 min/deployment  
*High Estimate*

Reduce Distribution Costs	Current	With IWOV	Savings
Number of content deployments each month	2,167	2,167	
Average minutes to deploy content to all target servers	20	1	19
Volume of content to be expired each month	181	181	
Average minutes to expire content	3	0	3
Average hourly fully burdened employee cost	\$36.06	\$36.06	
<b>Annual content deployment costs</b>	<b>\$316,406</b>	<b>\$15,625</b>	<b>\$300,781</b>

10 min/deployment  
*WAKM response*

Reduce Distribution Costs	Current	With IWOV	Savings
Number of content deployments each month	2,167	2,167	
Average minutes to deploy content to all target servers	10	1	9
Volume of content to be expired each month	181	181	
Average minutes to expire content	3	0	3
Average hourly fully burdened employee cost	\$36.06	\$36.06	
<b>Annual content deployment costs</b>	<b>\$160,156</b>	<b>\$15,625</b>	<b>\$144,531</b>

5 min/deployment  
*Low Estimate*

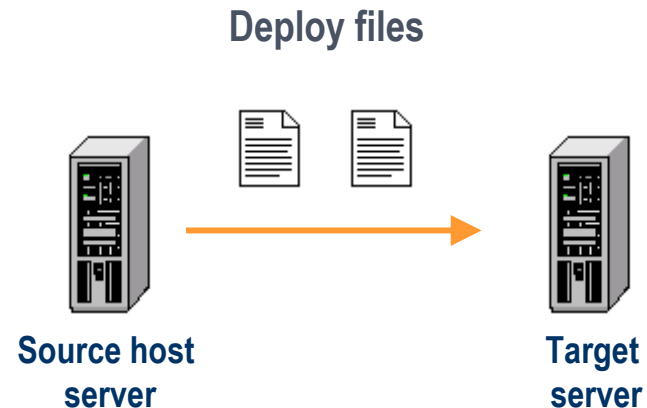
Reduce Distribution Costs	Current	With IWOV	Savings
Number of content deployments each month	2,167	2,167	
Average minutes to deploy content to all target servers	5	1	4
Volume of content to be expired each month	181	181	
Average minutes to expire content	3	0	3
Average hourly fully burdened employee cost	\$36.06	\$36.06	
<b>Annual content deployment costs</b>	<b>\$82,031</b>	<b>\$15,625</b>	<b>\$66,406</b>

# Supporting Data

- ◆ Data was provided by <person 1> and <person 2>, below are some key stats that guided my estimates.
- ◆ Person 1:
  - Distribute content to various sites about 4 to 5 times a week plus emergency changes
  - Approximately 30 minutes total to distribute content for dev, staging and production
  - Approximately 3 minutes to test each deployment
  - Main deployment techniques include FTP, secure FTP, rsync, FrontPage extensions, mapped drives – many are being deployed directly to live server
  - “Last I knew there were over 500 registered sites (tip of the iceberg)”
- ◆ Person 2
  - Average time to **create** content without IWOV was 125 minutes and with IWOV was 68 minutes
  - Average time to **deploy** content was at least 10 minutes without IWOV and up to 1 minute with IWOV (now a single click)

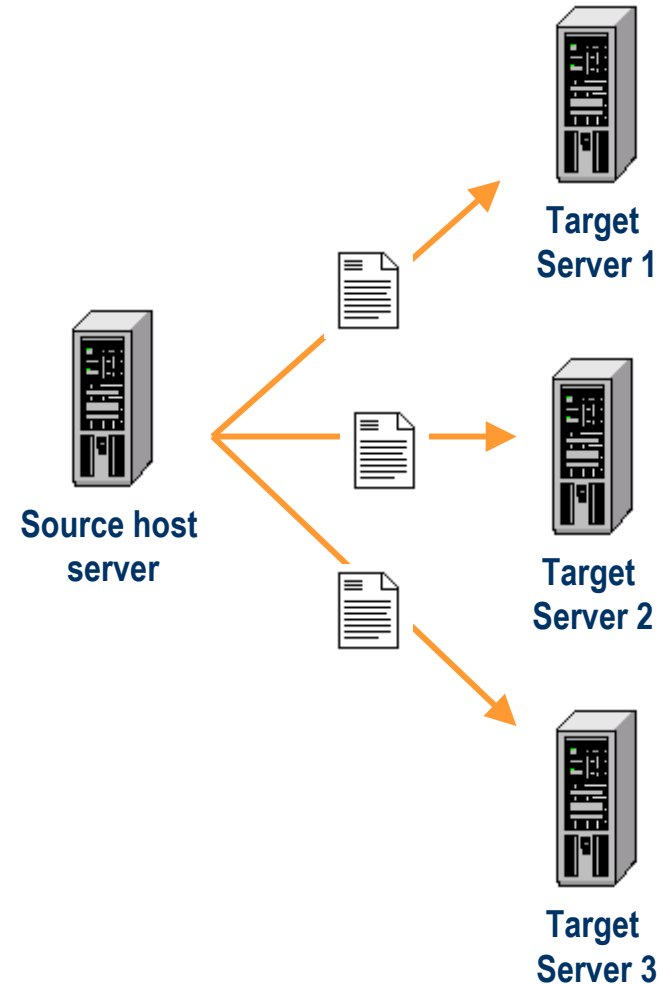
# How Deployment Works

- ◆ OpenDeploy base server is "source"
  - Typically installed on TeamSite server
- ◆ OpenDeploy receiver server is "target"
  - Typically installed on web server
- ◆ One base server can support any number of receivers
- ◆ Deployments can be multi-tier, transactional, and/or multi-target
- ◆ During deployment, OpenDeploy can filter out unwanted files, set permissions, and perform other automated tasks



# Using OpenDeploy to Promote Re-Use

- ◆ OpenDeploy can promote re-use by deploying common content from a single source to multiple receivers
- ◆ Thus, a "common" branch in TeamSite could support the development of shared content that is used by multiple different websites on different web servers
- ◆ Whenever an asset changes in the shared branch, it can be deployed to all required servers



## Commonly-sharable Assets

- ◆ Organizational artwork (logos, backgrounds, etc)
- ◆ Legal text (disclaimers, etc)
- ◆ Scripts and applets
- ◆ Organizational information (directories, department lists, contact information)

# Deployment Architecture Planning Issues

- ◆ High-level considerations for planning your deployment architecture
  - What are the source and target server platforms? Are they the same or different?
  - Will deployment involve crossing firewalls?
  - Will deployment be within a LAN or over a WAN? If deploying across a WAN, is the architecture consistent throughout?
  - Is encryption needed? If so, what level of encryption?
  - How many targets are there? Will any be used as intermediate hosts to deploy content to other targets?
  - Will the source files come from TeamSite areas or from the file system?
  - Do you want to deploy all files, or only changed files?

## Action Exercise: ROI and Deployment

- ◆ Within your group, spend 20 minutes discussing the following:
  - How do the ROI example time and cost estimates compare to your expectations?
  - What deployment implementation issues (firewall, security, etc) do you anticipate?
  - What, if any, deployment reuse opportunities can you foresee?
- ◆ After discussion, spend 30 minutes documenting the following:
  - Your group's 3 major areas of expected ROI improvement under ECM
  - A list of possible issues that you must analyze and plan for before implementing deployment
  - A list of deployment reuse opportunities
- ◆ Each group will then present their findings to the class one at a time

# Action Item Discussion

- ◆ Class presentation
- ◆ Questions
- ◆ Take a few minutes to consider other group action items
  - Integrate theirs with yours if needed



## TeamSite in the Enterprise

# Topic Objectives

- ◆ In this topic, we'll cover:
  - The architecture of the TeamSite server
  - How the TeamSite server relates to other enterprise components
- ◆ The action planning at the end of this topic will be:
  - Discussing agencies' plans for integrating TeamSite with other organizational servers

# TeamSite in the Enterprise

- ◆ The TeamSite server sits between your content developers and the production environment
  - Developers make their changes in TeamSite
  - TeamSite (and OpenDeploy) update the production environment
- ◆ TeamSite also integrates or collaborates with other server types:
  - Web servers
  - Application servers
  - Database servers
  - LDAP servers

# Virtualization

- ◆ **Virtualization** servers enable TeamSite to simulate the production environment for developers
  - HTML pages render, Java and JavaScript executes as if on the production site
- ◆ The goal: avoid the need to use artificial work-arounds during development
- ◆ When a developer views a file from inside TeamSite, the TeamSite server attempts to display the file to the developer in the same way the file would be displayed on a production site
- ◆ The TeamSite **proxy server** handles website virtualization

# How Virtualization Works: 1

- ◆ In the preceding diagram, did you note the 2 **virtualization** servers?
- ◆ TeamSite normally integrates with at least one **web virtualization server**
  - Such as iPlanet, IIS, Apache, etc
  - Typically the same type as the production web server being supported
  - This server renders the HTML for a virtualized page
- ◆ Often you will require an **application virtualization server**
  - Such as WebLogic, WebSphere, etc
  - This server executes server-side code for a virtualized page

## How Virtualization Works: 2

- ◆ The web and app virtualization servers can be:
  - Located on the same machine as TeamSite
  - Located on one or two other machines
- ◆ Typically:
  - The web virtualization server is installed on the same server as TeamSite
  - The app virtualization server *may* be co-located on the TeamSite server, but might also be separate

# Factors Affecting Virtualization Planning

- ◆ What OS is TeamSite running on?
- ◆ What OS are the production servers running on?
- ◆ What is the network connectivity between them?
- ◆ How many instances of the application virtualization server will be required?
  - Typical: one per active JSP programmer
- ◆ What types of files should be virtualized by the web server?
- ◆ Which should be virtualized by the app server?
- ◆ There are standard integration packages ("Turbos") for the most common application servers

## Action Exercise: Enterprise Integration

- ◆ Within your group, spend 15 minutes discussing the following:
  - What servers do you think your TeamSite server will need to integrate with?
  - What file types do you work with that can't be viewed with just a web server?
- ◆ After discussion, spend 10 minutes documenting the following:
  - A list of the servers you need to integrate or interact with from TeamSite
- ◆ Each group will then present their findings to the class one at a time

# Action Item Discussion

- ◆ Class presentation
- ◆ Questions
- ◆ Take a few minutes to consider other group action items
  - Integrate theirs with yours if needed



## TeamSite Branching

# Topic Objectives

- ◆ In this topic, we'll cover:
  - TeamSite branching patterns
  - Branching and virtualization issues
  - Locking models
  - Using branches to implement re-use and sharing
- ◆ The action planning at the end of this topic will be:
  - Designing your group's branching model

# Branching Patterns

- ◆ Branches as projects
- ◆ Branches as sub-projects
- ◆ Branches for teams
- ◆ Branches for common assets
- ◆ Branches for different deployment targets

## Branch Concepts Review

- ◆ Branches are content containers
- ◆ Each branch is a separate container
- ◆ Each branch has a staging area, one or more editions, and zero or more workareas
- ◆ Each backing store has a **main** branch
- ◆ **Best practice:** Don't put content in **main**; use **main** as a "mount point" for your organization's primary branches
- ◆ This allows each primary branch to have separate owners and groups for shared access
- ◆ **Best practice:** create special user account(s) for branch ownership—don't assign ownership to personal user IDs

# Branching and Virtualization

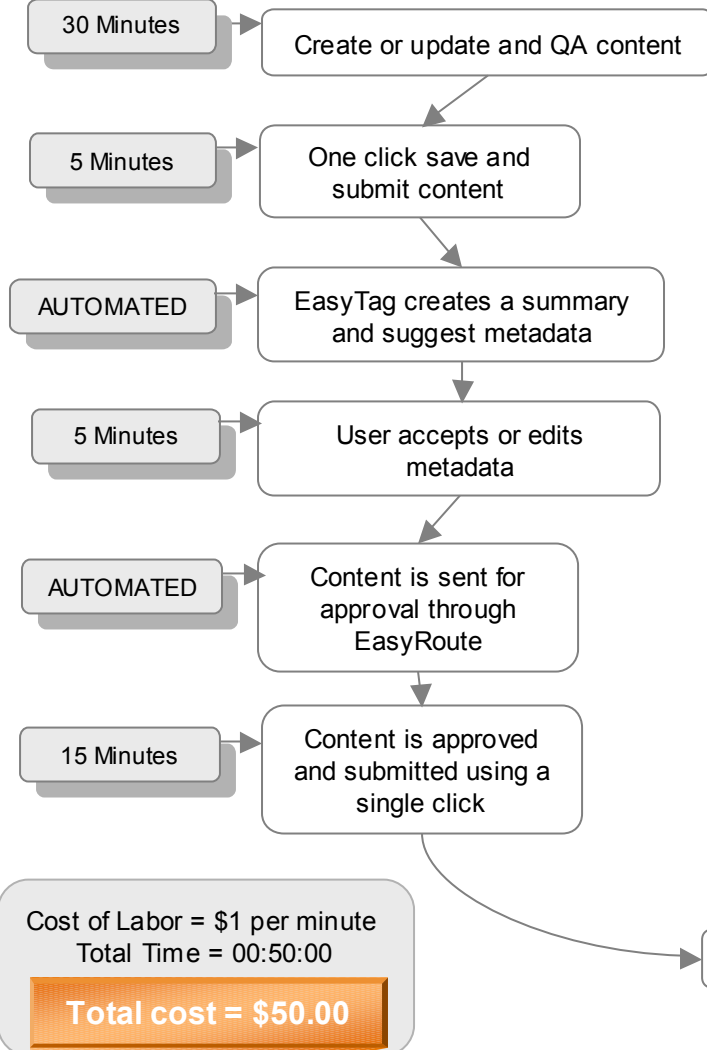
- ◆ If a page viewed in TeamSite includes assets from more than one branch, the TeamSite server must be configured to properly render the page
  - Default: TeamSite assumes all included/linked assets are in the current TeamSite area
  - Required: Make TeamSite "smart", so it knows that images are in the **main/images** branch, styles are in the **main/common** branch, etc
- ◆ This "intelligence" is configured using **proxy rules**
- ◆ Proxy rules only affect the development environment
- ◆ Proxy rules take time to create and test

# Considerations for Determining Branch Structure

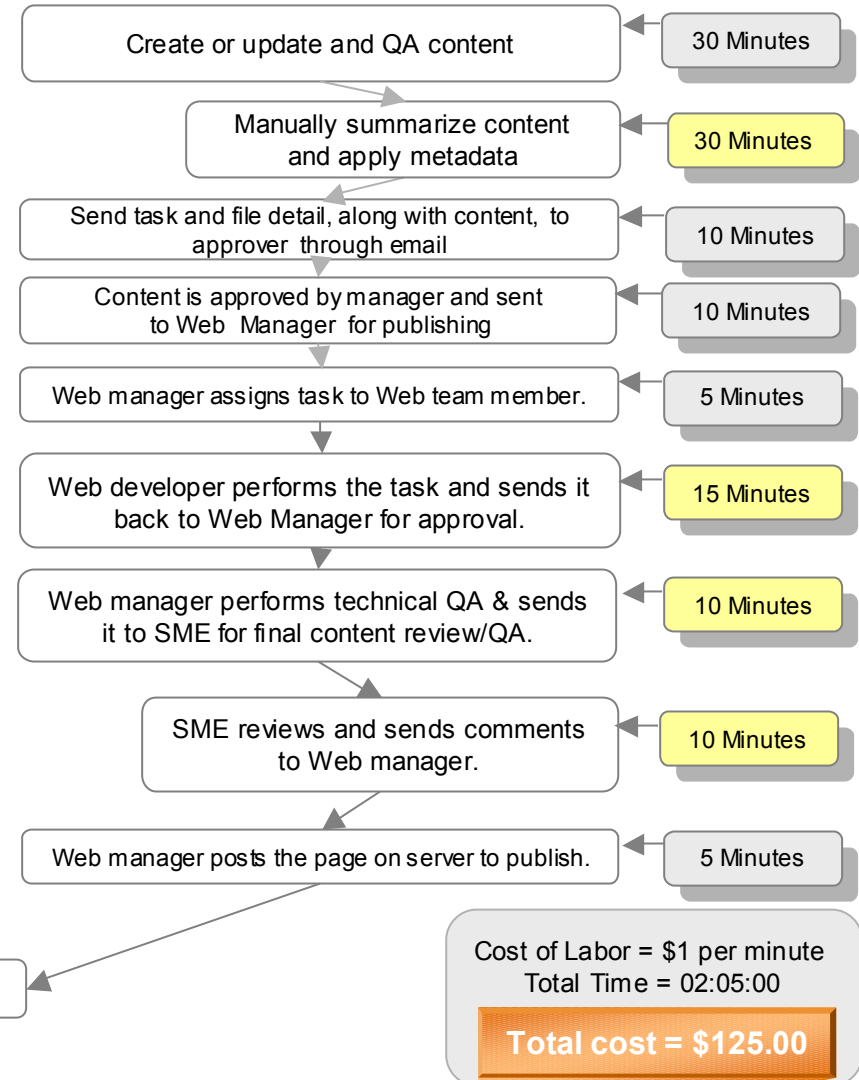
- ◆ Logical division
  - Aggregate assets based on type or use
- ◆ Update frequency
  - Aggregate assets that are updated together
- ◆ Shared development environment
  - Aggregate assets that are shared between teams
- ◆ Workarea configuration
  - Within each branch, how will users access the content?
- ◆ Deployment to production
  - How will the content be delivered from the development environment (TeamSite) to the production environment (web servers)?

# Process Improvement with Collaborative Doc Mgmt

## With IWOV TeamDoc

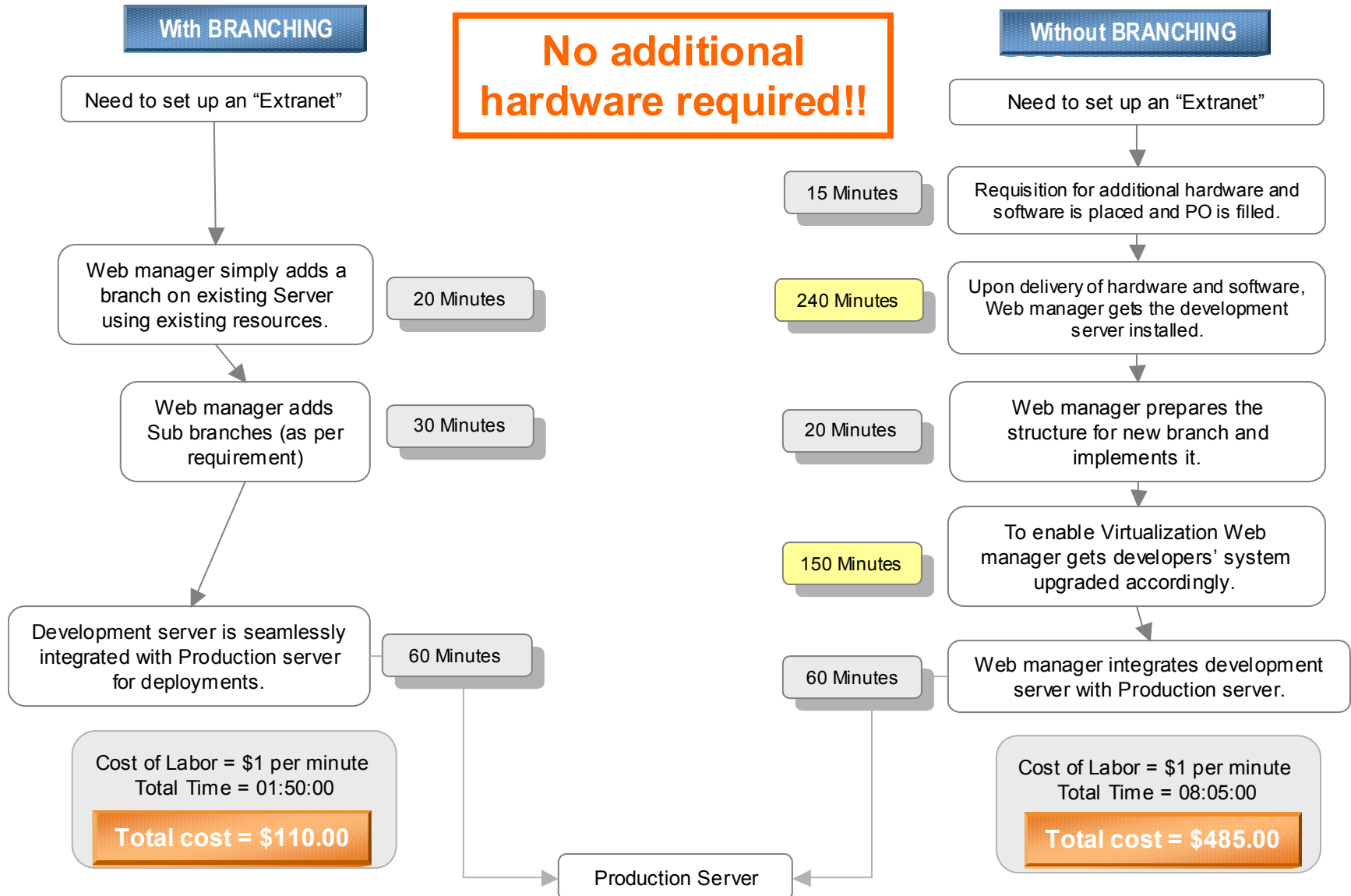


## Without IWOV TeamDoc



# Process Improvement with Branching

**No additional hardware required!!**



# Branch Lock Mode

- ◆ Default: submit locking (preferred)
- ◆ Alternatives
  - Optional write locking
  - Mandatory write locking
- ◆ Write locking negatively impacts collaboration, flexibility
- ◆ **Best practice:** use submit locking unless some technical or business requirement makes it necessary to use write locking

# Identify Common Assets

- ◆ Types of common web assets
  - Images
  - Standard text
  - Style sheets
  - Scripts
- ◆ One approach: use a "common" branch for assets that can be shared between teams
- ◆ Issue: who (ie, what team) "owns" the common branch?

# Planning for Branch Security

- ◆ Team branches administered by a team member
- ◆ Team branches read-only for that team (plus system administrators/masters)
- ◆ Common branches must be readable by all client teams
  - For this you need a UNIX group that includes all possible users of that branch
- ◆ Branches can be "hidden" from those who don't have access to them

# Planning for Asset Security

- ◆ Each asset has its own access permissions
- ◆ Asset permissions affect how the team works with that asset
- ◆ To be modifiable, an asset must:
  - Be in a workarea a user has access to (owner or group)
  - Have a writable permission for that user or a group the user belongs to
- ◆ **Best practice:** Submit filtering (via **submit.cfg**) can automate the enforcement of desired development asset permissions
- ◆ Must first identify types of assets (ie, .html, .jsp, .java, .class, etc, then determine what groups require access to that asset type

## Planning for Re-Use

- ◆ Goal: implement reusable content so that it can be maintained as a single asset
- ◆ Re-usable assets are implemented in more than one location on a website
- ◆ Can be an entire page or a part of a page
- ◆ Can be visible, or hidden
- ◆ Might be shared between teams (agencies), or might be restricted to one team's content

# Planning for Sharing

- ◆ Identify cross-agency assets
- ◆ Identify cross-agency template types
- ◆ Will teams share their branches with other teams,  
or
- ◆ Will common assets be stored in a common branch?
- ◆ Shared content in another branch is normally accessed through the branch's STAGING area, so workareas are not required for others to access the shared content

# MultiStore

- ◆ A single TeamSite server can have up to 8 active backing stores
- ◆ Each store can reside on separate disk volumes, can be managed and backed up individually
- ◆ Maintenance on one backing store affects only the team(s) using branches in that backing store
- ◆ Disk hardware problems can be isolated to minimize impact
- ◆ Use separate backing stores for each agency (or collections of agencies?)
- ◆ Additional cost

# System Configuration Branch

- ◆ **Objective:** use TeamSite to version-control its own configuration files
- ◆ **Advantage:** recoverable configuration, easier to access configuration files
- ◆ **Procedure:**
  1. Create a branch named "config" (or similar)
  2. Copy in contents of *iw-home/local* + *iw.cfg* (and any other system configuration files desired)
  3. Set up deployment as final step of submit process for this branch
  4. Do all system configuration edits here, and, when done, deploy will update the real files

# Deployment Issues

- ◆ If more than 1 branch contains content for a particular web server, how, when and where is it "reassembled" into one logical whole?
  - In TeamSite: an **integration** branch
  - During deployment: **selective deployment targeting**

## Sub-Branching

- ◆ Within a team branch, the team may choose to create additional sub-branches
- ◆ Each sub-branch might contain content being worked as a separately controlled project, with its own development and release schedule
- ◆ Each sub-branch can contain its own editions, workareas, and security settings
- ◆ Sub-branches can be long-term or short-term in nature
  - Long-term/permanent branches usually represent a decomposition of structure based on function or ownership
  - Short-term branches represent a limited-effort project, such as a new release, etc

# Workareas

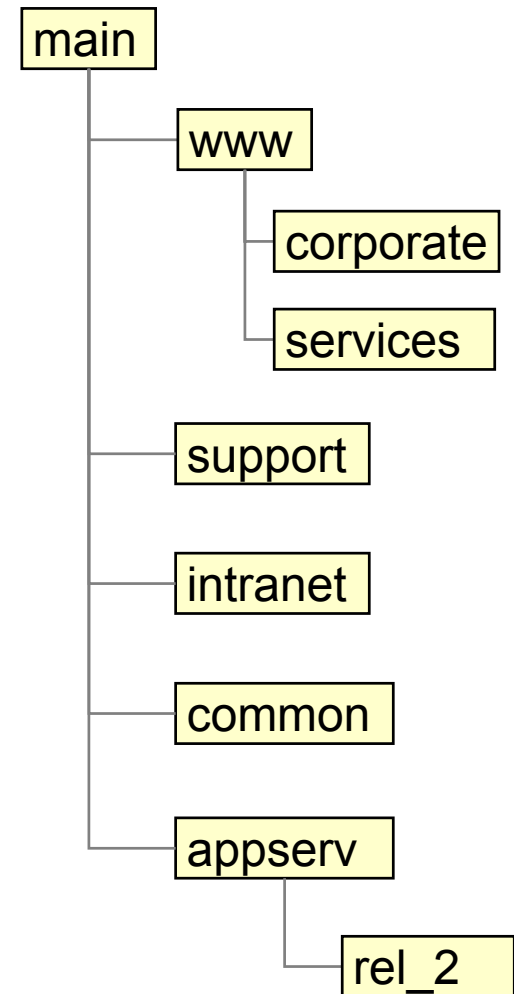
- ◆ Within a branch, workareas enable access to modify content
- ◆ Workareas can be permanent or temporary
- ◆ Workareas can be used by individuals, groups, or automated processes such as workflows
- ◆ Each team should have a TeamSite Administrator, who manages workareas and sub-branches for the team
- ◆ Reviewers/approvers will require access to workareas too
- ◆ **Best practice:** Even if a workarea is intended for an individual, make sure it is shared with your system administration group

# Signs of a Good Branching Scheme

- ◆ Everyone knows where their assets are and can access them quickly
- ◆ New teams or projects are easy to integrate
- ◆ Shared assets can be updated in one location and automatically apply wherever they are in use
- ◆ Nobody can access or modify restricted content
- ◆ Each team member only sees the branches and/or workareas they need to use (reduced clutter)
- ◆ The scheme is as simple as possible
- ◆ Changes in personnel don't require changes to branch ownership
- ◆ Modifications or maintenance on one branch affect a minimum of the other teams

## Example Branch Pattern

- ◆ **www** is the corporate external web site
  - Decomposed into 2 specialized sub-branches by functional area
- ◆ **support** is a customer-support site
- ◆ **intranet** is a private, internal website
- ◆ **common** stores assets used in more than one branch
  - Requires a proxy rule for correct virtualization
- ◆ **appserv** stores J2EE application server code that is called from various other pages on all sites
  - Also requires proxy server rules
  - Has short-term **rel\_2** sub-branch for new release being developed



## Action Exercise: Branch Design

- ◆ Within your group, spend 30 minutes discussing the following:
  - What primary branches will you need?
  - What secondary or sub-branches will you need?
  - What type of locking model makes the most sense for your business model?
- ◆ After discussion, spend 20 minutes documenting the following:
  - A tree diagram of your branches
  - Identify for each branch:
    - Locking model
    - Who will use it
    - What content it contains
- ◆ Each group will then present their findings to the class one at a time

# Action Item Discussion

- ◆ Class presentation
- ◆ Questions
- ◆ Take a few minutes to consider other group action items
  - Integrate theirs with yours if needed

# End of Session

- ◆ This concludes today's session
- ◆ Next session: **December 18, 2002**
  - TeamSite Security
  - Workflow Scenarios
  - Designing a Workflow